REMARKS

By this amendment, Claims 42-44, 48-49, 51-53, and 57-58 are amended, Claims 44-45 and 54-55 are canceled, and Claims 60-61 are added. No new matter has been added.

Each issued raised in the Office Action mailed October 28, 2008 ("Office Action") is addressed hereinafter.

CLAIM REJECTIONS – 35 U.S.C § 103

Claims 42-49 and 51-58 were rejected under 35 U.S.C. § 103(a) as allegedly unpatentable over *Chakravarthy, et al.* ("Composite Events for Active Databases: Semantics, Contexts and Detection", Proc. Of the 20th VLDB Conf., Santiago, Chile © 1994, pp. 606-617 hereafter referred to as "Chakravarthy") in view of Etzion, et al., (US Patent on No. 6,604,903) (hereinafter "*Etzion*").

Claims 50 and 59 were rejected under 35 U.S.C. § 103(a) as allegedly unpatentable over *Chakravarthy* and *Etzion* and further in view of Kumar, et al., (US Patent on No. 7,149,738).

CLAIM 42

Claim 42 recites:

> A machine-implemented method for managing event-condition-action rules in a database system, the method comprising the computer-implemented steps performed by said database system of:
>
> storing, in a database managed by said database system, rule data that defines a composite event comprised of two or more primitive events, at least one condition related to the composite event, and at least one action to be performed upon satisfaction of said at least one condition;
>
> detecting a first database event as an occurrence of a first one of the primitive events;

determining whether the first database event satisfies a first sub-condition of said at least one condition, wherein said rule data indicates that satisfaction of said first sub-condition is not sufficient to satisfy said at least one condition;

persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event;

detecting a second database event as an occurrence of a second one of the primitive events;

reading said results data from said database;

determining whether the at least one condition is satisfied based on the results data read from the database and the second database event.

Applicants respectfully submit that at least the above-bolded features of Claim 42 are not disclosed or in any way rendered obvious by *Chakravarthy* and *Etzion*.

Support for the amendments to Claim 42 can be found throughout the specification taken as a whole and, in particular, in paragraph [0006] (discussing problems associated with conventional ECA processing), paragraph [0028] (stating "The results from the incremental evaluation of related primitive events are stored persistently in the database."), paragraph [0087] (stating "Blocks 206 and 208 can be repeated for each primitive event occurrence associated with a composite event structure, **with results of each primitive event persistently stored in the database** at least until all of the other sibling primitive events are processed or until the primitive event expires according to a rule set property, such as a duration policy." (Emphasis added.)), and paragraph [0088] (stating "Furthermore, because the database-enabled rules engine supports composite events and the **persistent storage of incremental evaluation of conditions with respect to primitive events** that make up a composite event, there is no restriction on the size of rule sets or the number of events that can be processed." (Emphasis added.)).

Claim 42 has been amended to more fully clarify what Applicants' regard as their invention and not for the purpose overcoming the cited art. For example, Claim 42 now makes

clear that determining whether the condition related to the composite event is satisfied is based on results data **read from the database**; the results data having previously been **persistently stored in the database**. The persistently stored results data indicating that a first database event satisfied a sub-condition of the condition related to the composite event. By performing the method of Claim 42, a database system can avoid problems associated with certain memory-based data structures for processing composite events. In particular, implementations that employ such memory-based data structures such as some middleware applications do not scale well for composite events with large sets of conditions and large numbers of primitive events because the results of evaluating parts of the composite event as one or more the primitive events occur must be held in memory at least until a condition related to the composite event is satisfied. Thus, such implementations are bounded by the size of physical memory.

In contrast, by "persistently storing **in the database** results data that indicates that said first sub-condition was satisfied by said first database event", the approach of Claim 42 is not bounded by the size of physical memory, but by the size of the database which can be much larger than physical memory. The approach of Claim 42 is not taught or in any way suggested by *Chakravarthy* and *Etzion*.

ANALYSIS OF THE CITED ART

The Office Action at page 5 contends that it would have been obvious to one skilled in the art to apply the teachings of *Etzion* for the benefit of *Chakravarthy*. Specifically, the Office Action contends that it would have been obvious to apply the teachings of *Etzion* at col. 11, line 61 – col. 12, line 5 which the Office Action alleges teaches "persistently storing the results of the determining in the database" for the benefit of *Chakravarthy* at page 615 which the Office Action alleges teaches "determining whether the at least one condition is satisfied based on the

persistently stored result and the second database event". Applicants respectfully submit that for the reasons provided hereinafter the proposed combination of *Etzion* and *Chakravarthy* does not satisfy Claim 42.

1. *Etzion* does not teach or suggest "persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event".

Etzion describes a "situation awareness system" which employs a candidate list data structure into which the situation awareness system maps the event instances it receives. (*See Etzion*, col. 13, lines 35-49 and figure 3.) Specifically, a candidate list is instantiated "for every situation that is included in the rules defined at step 40." (Id.) "Each [event] instance that meets the criteria of a given situation is a candidate 94 to be part of the situation to which the system is to respond." (Id.) "The candidates are stored in this [candidate list] structure in order to identify and preserve their order for the purposes of **processing**." (*Id.*) (Emphasis added.) This "processing" includes determining 'whether the instances that have been entered in the candidate list satisfy the requirements of the operators that define the complex event, **as well as meeting the remaining "where" conditions that define the situation to which system 20 is required to react**." (*Etzion*, col. 17, lines 10, 15) (Emphasis added.)

Etzion appears to employ a memory-based data structure which suffers from the same problems as those discussed in the background of Applicants' specification. In particular, *Etzion's* candidate list would not scale well for a situation with a large set of candidates because the candidate list would be bounded by the amount of available physical memory to store the candidates.

Further, nothing in *Etzion* teaches or suggests that the candidate list is persistently stored in a database. While *Etzion* does describe a general-purpose computer complete with a memory

and a database, *Etzion* does not in any way describe persistently storing the candidate list data structure in the database. (*See Etzion*, col. 7, line 45 – col. 8, line 6 and figure 1.) Indeed, the candidate list data structure of *Etzion* is a linked-list data structure well-suited efficient representation within the volatile physical memory of general purpose computer. (*Id.*) Given the situation awareness system's need to manipulate the candidate list for each event instance received, one skilled in the art would not reasonably read *Etzion* to describe persistently storing the candidate list in a database during event processing.

Moreover, *Etzion* does not describe "persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event", as featured in Claim 42. In particular, there is nothing in *Etzion* that describes **persistently storing in a database** the results of determining whether a candidate in the candidate list of *Etzion* satisfies a "where" condition that defines the situation to which situation awareness system is required to react. From a thorough examination of *Etzion*, Applicants can discern only that *Etzion* determines whether instances that have been entered into the candidate list meet the "where" conditions that define the situation to which the candidate list corresponds; but Applicants cannot discern anything in *Etzion* that would suggest that the results of these determinations are persistently stored in a database. Thus, Applicant respectfully submits that *Etzion* does not teach or suggest "persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event", as featured in Claim 42.

The portion of *Etzion* cited in the Office Action describes the semantics of certain situation definitions involving complex events but says nothing about persistently storing the results of evaluating "where" conditions in a database. For example, given the complex event operator "before(first e1, each e2)" *Etzion* notes that event instances of type e1 and type e2 that satisfy the operator can be "consumed" as they occur. Thus, given the event instances provided

in Table V of *Etzion*, when event instances are consumed, the operator "before(first e1, each e2)" returns "(e11, e21), (e12, e22) and (e13, e23)" instead of "(e11, e21), (e11, e22), and (e11, e23)".

Etzion also notes that event instances that are not consumed can be discarded rather than "saved for subsequent use." (*Etzion*, col. 12, lines 4-6) However, saving for subsequent use means nothing more than storing the event instance in the candidate list data structure of *Etzion* which, as explained above, is not persistently stored in a database during event processing. (*See also*, *Etzion*, col. 16, lines 40 – 57 (describing "override conditions" related to the "first" and "last" quantifiers that are applied to event instances to "determine whether [the event instances] should be added to the new or existing candidate list.").) Thus, storing an event instance in the candidate list data structure of *Etzion* for "subsequent use" is not "**persistently storing** in the database **results data that indicates that said first sub-condition was satisfied by said first database event**" as featured in Claim 42.

Based on the foregoing, Applicants respectfully submit that *Etzion* does not teach or suggest "persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event" as alleged in the Office Action.

2. *Chakravarthy* does not overcome the deficiencies of *Etzion*.

Chakravarthy describes "parameter contexts" for use in detection of composite events. Indeed, the parameter contexts of *Chakravarthy* appear to be quite similar to the "Initiators" and "Terminators" of *Etzion*. Both used sued for efficient management and detection of composite events.

However, also like *Etzion*, *Chakravarthy* employs a physical memory-based data structure for composite event detection. In particular, *Chakravarthy* states "[t]he local composite event detector and the application share the same address space and our event detector uses an

event graph similar to operator trees." (*See* Chakravarthy, § 5.3 and Figure 4.) Further, *Chakravarthy* states "[a] linked list that contains the parameters of each primitive event (as a list) that participates in the detection of the composite event is computed and passed to the rule associated with that event." Nothing in *Chakravarthy* suggests that the "operator trees" or the "linked list" is stored **persistently** in a database.

In fact, modifying *Chakravarthy's* "Sentinel Architecture" as proposed in the Office Action would render the Sentinel Architecture unsatisfactory for at least one of its intended purposes which is to provide a "clean separation" from composite event detection and primitive event detection. Specifically, in *Chakravarthy's* Sentinel Architecture, the composite event detector is implemented as an application module **separate from the database** because as expressly stated in *Chakravarthy*, a "clean separation of the detection of primitive events (**as an integral part of the database**) from that of composite events allows one to: i) implement a composite event detector as a separate module (as has been done) and ii) introduce additional event operators without having to modify the detection of primitive events." (Id.) (Emphasis added.) Thus, modifying *Chakravarthy's* "Sentinel Architecture" to perform composite event detection within the database would not permit the introduction of additional event operators without having to modify the detection of primitive events.

3. Claims 42 and 51 are patentable over *Chakravarthy* and *Etzion*.

Based on the foregoing, Applicants respectfully submit that *Chakravarthy* and *Etzion*, either individually or in combination, does not teach or suggest at least the following bolded features of Claim 42 and the proposed modification of *Chakravarthy* would render *Chakravarthy*'s Sentinel Architecture unsatisfactory for at least one of its intended purposes. Consequently, Claim 42 is patentable over *Chakravarthy* and *Etzion*.

persistently storing in the database results data that indicates that said first sub-condition was satisfied by said first database event;

detecting a second database event as an occurrence of a second one of the primitive events;

reading said results data from said database;

determining whether the at least one condition is satisfied based on the results data read from the database and the second database event.

Claim 51 recites similar limitations to those recited in Claim 42 and is allowable over *Chakravarthy* and *Etzion* for the same reasons.

REMAINING CLAIMS

The pending claims not discussed so far are dependant claims that depend on an independent claim that is discussed above. Because each dependant claim includes the features of claims upon which they depend, the dependant claims are patentable for at least those reasons the claims upon which the dependant claims depend are patentable. Removal of the rejections with respect to the dependant claims and allowance of the dependant claims is respectfully requested. In addition, the dependent claims introduce additional features that independently render them patentable. Due to the fundamental differences already identified, a separate discussion of those features is not included at this time.

CONCLUSIONS

For the reasons set forth above, it is respectfully submitted that all of the pending claims are now in condition for allowance. Therefore, the issuance of a formal Notice of Allowance is believed next in order, and that action is most earnestly solicited.

The Examiner is respectfully requested to contact the undersigned by telephone if it is believed that such contact would further the examination of the present application.

Please charge any shortages or credit any overages to Deposit Account No. 50-1302.

Respectfully submitted,

Hickman Palermo Truong & Becker LLP

Date: January 28, 2009

/AdamCStone#60531/
Adam Christopher Stone
Reg. No. 60,531

2055 Gateway Place, Suite 550
San Jose, California 95110-1083
Telephone No.: (408) 414-1231
Facsimile No.: (408) 414-1076